NESNEYE TABANLI PROGRAMLAMA - II
Kalıtım

Öğr. Gör. Murat ASLANYÜREK

Kırklareli Üniversitesi

Pınarhisar MYO

KALITIM KAVRAMINA GİRİŞ

- Nesneye yönelik programlamanın temel taşlarından biri olan **Kalıtım**, sınıflardan yeni sınıflar türetmeyi sağlar. Türetilen bu yeni sınıflar, türetildiği sınıfların özelliklerine sahip olur. Örneğin; **Araç** adında bir sınıfımız olsun. **Otomobil** sınıfını bu **Araç** sınıfından türettiğimiz zaman **Otomobil** sınıfı, **Araç** sınıfının özelliklerini taşır. Aynı şekilde **Hayvan** adında bir sınıfımız olsun. **Sürüngenler** adında bir sınıfı bu **Hayvan** sınıfından türetirsek, **Hayvan** sınıfını genişletmiş oluruz ve Sürüngenler sınıfı, Hayvan sınıfının özelliklerini taşımış olur. Fakat bunun tersi doğru değildir. **Hayvan** sınıfı, **Sürüngenler** sınıfının her özelliğini taşımayabilir.
- Kalıtım yapmak için kullanacağımız kelime **extends**'dir.

```
public class Hayvan
{
}
public class Surungenler extends Hayvan
{
}
```

- Bu şekilde tanımlama yaptığımızda Surungenler Sınıfındaki elemanlar, Hayvan sınıfının elemanlarını miras almış oluyor. Hayvan sınıfından **extends** ile bir sınıf daha oluşturabiliriz, fakat bir sınıfı hem Hayvan hem de Memeliler sınıfından türetemeyiz. Yani bir sınıf, birden fazla sınıftan türetilemez.

```java
package Kalitim;

public class Hayvan {
    public void nefesAl(){
        System.out.print("Nefes Alma Metodu");
    }
}

```

```java
package Kalitim;
    // Surungen sınıfını Hayvan sınıfından türettik
 class Surungen extends Hayvan {

    public static void main(String[] args){
        Surungen s = new Surungen();
        // artık bu sınıfımız da nefesAl metoduna sahip oldu
        s.nefesAl();
    }
}
```

Console | Problems | Debug Shell | Search

<terminated> Surungen [Java Application] C:\Program Files\Java\jdk-13.0.1\bin\javaw.ex

```
Nefes Alma Metodu
```

- Burada Surungen sınıfını Hayvan sınıfından türettik ve Surungen sınıfı içerisinde, nefesAl () metodunu tanımlamadan bu metoda erişebildik.

- ***Not:*** *Kalıtımın en büyük avantajı üst sınıftaki metod ve değişkenleri tekrar yazmaktan kurtarmasıdır.*

SUPER CLASS (ÜST SINIF) VE SUB CLASS (ALT SINIF) KAVRAMLARI

- **Otomobil** adında bir sınıfımız olsun. **Yarış otomobili** ve **aile otomobili** sınıfları, **Otomobil** sınıfının **alt sınıflarıdır(Sub-Class). Otomobil** sınıfı da bu **sınıfların üst sınıfıdır(Super-Class).**
- Otomobil örneğinde alt sınıflar, üst sınıfının tüm özelliklerine sahiptir. Bir yarış otomobili, otomobil sınıfındaki vites ve hız gibi değişkenlere ve bunun gibi metodlara sahip olur. Bunların yanında kendine ait metod ve değişkenleri de olabilir. Bir yarış otomobilinde normal otomobilden farklı olarak ek özellikler de bulunabilir. Bu ek özellikleri kendi sınıfına dahil edebilir. Bunun dışında yarış otomobili sınıfı, Otomobil sınıfındaki freneBas adlı bir metodu değiştirebilir. Buna metod overriding denir.

```java
package Kalitim;

 class Telefon {
    public Telefon(){
        System.out.println("Telefon sınıfının Yapıcısı");
    }
}
class AkilliTelefon extends Telefon{
    public AkilliTelefon(){
        System.out.println("Akıllı Telefon Yapıcısı");
    }
}
class CepTelefonu extends Telefon{
    public CepTelefonu(){
        System.out.println("Cep Telefonu Sınıfının Yapıcısı");
    }
}
class Nokia extends CepTelefonu{
    public Nokia(){
        System.out.println("Nokia Sınıfının Yapıcısı");
    }
}
```

```java
package Kalitim;

public class Calistir_Telefon {
    public static void main(String[] args){
        Nokia n1 = new Nokia();
    }
}
```

Console | Problems | Debug Shell | Search

```
<terminated> Calistir_Telefon [Java Application] C:\Program Files\J
Telefon sınıfının Yapıcısı
Cep Telefonu Sınıfının Yapıcısı
Nokia Sınıfının Yapıcısı
```

- Bu örneğimizde Calistirma sınıfındaki main metodunda, Nokia sınıfından bir nesne ürettik. Nokia Sınıfı, CepTelefonu sınıfından türetilmiştir. CepTelefonu sınıfı ise Telefon sınıfından türetilmiştir. En yukarıdan başlayarak yapıcılar çağırılır ve en sonda ise ilk oluşturulan nesnenin olduğu sınıfın yapıcısı çağırılır. *Bir sınıftan nesne oluşturulduğunda, o sınıfın yapıcısından önce, türetildiği sınıfın yapıcısı çağırılır.* Bu şekilde en üst seviyedeki Super Class'ın yapıcı metodu çağırılır.

```java
package Kalitim;
    class Sinif1{
        int x,y;

        void degerAta(int a, int b){
            x=a;
            y=b;
        }
    }
    class Sinif2 extends Sinif1{

        int carp(){
            return x*y;
        }
    }
public class Kalitim {
    public static void main(String[] args){
        Sinif2 s2= new Sinif2();
        s2.degerAta(5, 9);
        System.out.println(s2.carp());
    }
}
```

Console | Problems | Debug Shell | Search

<terminated> Kalitim [Java Application] C:\Program Files\Java\jdk-

45

- Sinif2'yi Sinif1'den türettik ve main metodunda Sinif2'den oluşturduğumuz nesne ile **Super Class** olan Sinif1'in degerAta metodunu çağırdık. Bu metoda 5 ve 9 değerlerini gönderdik. Bu parametreler, x ve y değişkenlerine atandı. Daha sonra s2 nesnesi üzerinden carp metodunu çağırdık. Bu metod da, değerlerini atadığımız x ve y değişkenlerini çarparak geriye sonucu döndürdü.
- Peki, bu şekilde türetilen bir sınıf, türediği sınıfın her elemanına erişebilecek mi?

- Eğer **Super class**'da **private** elemanlar varsa **sub class**'lardan bu elemanlara erişilemez.

```java
package Kalitim;
class Sinif3{
    private int sinif=2;
    public String birim="PMYO";
    String ders="NTP-2";

}
class Sinif4 extends Sinif3{
    public String program="Bilgisayar Prog.";
}

public class Kalitim2 {
    public static void main(String[] args){
        Sinif4 nesne = new Sinif4();
        //System.out.println(nesne.sinif); //Erişemez
        System.out.println(nesne.birim); //Erişir
        System.out.println(nesne.program);
        System.out.println(nesne.ders);
    }
}
```

Console | Problems | Debug Shell | Search

<terminated> Kalitim2 [Java Application] C:\Program Files\Java\jdk-13.0.1\bin\jav

```
PMYO
Bilgisayar Prog.
NTP-2
```

SUPER METODU İLE ÜST SINIFIN YAPICISINI ÇAĞIRMAK

- Türetilen sınıf, türeyen sınıfın yapıcısına erişmek isterse **super** anahtar kelimesi kullanılır.
- A sınıfından B sınıfını türetmiş olalım. B sınıfının yapıcısının içerisinde **super()** metodunu çağırırsak, A sınıfın yapıcısı çağrılmış olur.
- **Not:** ***super()*** *metodu ilk sırada yazılmalıdır.*

```
1  package Kalitim;
2      class A{
3          public A(){
4              System.out.println("A'nın yapıcısı");
5          }
6      }
7
8  public class B extends A{
9      public B(){  // B nin yapıcısı
10
11         super(); // A'nın yapıcısı çağırıldı
12         System.out.println("B'nin yapıcısı");
13     }
14     public static void main(String[] args){
15         B nesne = new B();
16     }
17 }
```

Console ⊠ Problems Debug Shell Search

<terminated> B [Java Application] C:\Program Files\Java\jdk-13.0.1\bin\jav

```
A'nın yapıcısı
B'nin yapıcısı
```

```java
package Kalitim;
class A2{
    public A2(){
        System.out.println("A'nın yapıcısı");
    }
    public A2(int a){
        System.out.println("A'nın parametreli yapıcısı");
    }
}

public class B2 extends A2{
    public B2(){  // B nin yapıcısı
    //super();
    super(2); // A'nın parametreli yapıcısı çağırıldı
    System.out.println("B'nin yapıcısı");
    }
    public static void main(String[] args){
        B2 nesne = new B2();
    }
}
```

Console | Problems | Debug Shell | Search

<terminated> B2 [Java Application] C:\Program Files\Java\jdk-13.0.1\bin\javaw.exe

```
A'nın parametreli yapıcısı
B'nin yapıcısı
```

- A sınıfımızda 2 adet constuctar (yapıcı) var. Biri parametresiz diğersi ise 1parametre alıyor.
- **super** kullanılarak parametreli yapıcı çağrılıyor.

- **super** anahtar kelimesi ile sınıftaki değişkenlere erişebiliriz.

- super anahtar kelimesi this anahtar kelimesine benzer. Farkı super anahtar kelimesiyle üst sınıftaki elemanlara erişebilirken this anahtar kelimesiyle içinde bulunduğu sınıfın elamanına erişilebilir.

```java
package Kalitim;

class UstSinif{
    int x;
}

public class AltSinif extends UstSinif{
    int x;
    public AltSinif(int i, int j) {
        super.x=i; // UstSinifin x değişkenine atar
        x=j; // AltSinifin x değişkenine atar

        System.out.println(super.x);
        System.out.println(x);

    }

}
```

```java
package Kalitim;

public class superKullan {
    public static void main(String[] args){
    AltSinif nesne = new AltSinif(6, 9);
    }
}
```

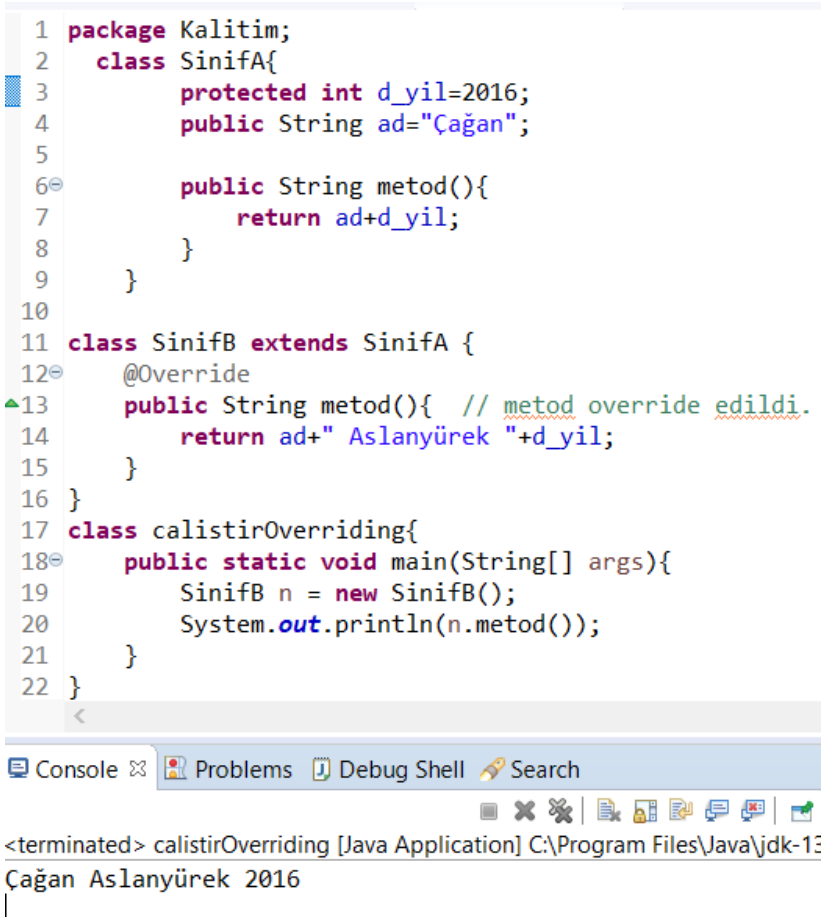
Console | Problems | Debug Shell | Search

<terminated> superKullan [Java Application] C:\Program Files\Java\

```
6
9
```

METOD OVERRIDING

- Kalıtım konusunda anlattığımız gibi, türetilen sınıf türediği sınıfın tüm özelliklerini taşıyor ve kullanabiliyordu. Fakat miras aldığı bu metodları aynen kullanmak istemezse, ne yapmamız gerekir? Burada overriding kavramı karşımıza çıkıyor. Bir sınıf, türetildiği sınıftaki bir metodu değiştirmek istediği zaman bunu overriding ile gerçekleştiriyoruz. Bu şekilde metodları override ettiğimiz zaman alt sınıf, üst sınıfın metodunu doğrudan kullanmak yerine kendine ait bir metodu oluyor.



```java
package Kalitim;
  class SinifA{
        protected int d_yil=2016;
        public String ad="Çağan";

        public String metod(){
            return ad+d_yil;
        }
    }

class SinifB extends SinifA {
    @Override
    public String metod(){  // metod override edildi.
        return ad+" Aslanyürek "+d_yil;
    }
}
class calistirOverriding{
    public static void main(String[] args){
        SinifB n = new SinifB();
        System.out.println(n.metod());
    }
}
```


- Bu örnekte SinifB, SinifA'nın metodunu override ediyor. Artık kendine ait bir metodu oldu. main metodunda bu SinifB içerisindeki metodu çağırdığımızda, SinifA'nın değil SinifB'nin metodu çağırılıyor. Eğer SinifB içerisinde bir metod tanımlamasaydık, doğrudan SinifA içerisindeki metodu çalıştıracaktı.

- **Not:** *Bir metod override edilebilmesi için belli başlı kurallar vardır. Her metod override edilemez.*
  - *Override edilecek metodların dönüş tipi, metod adı, parametre listeleri aynı olmalıdır.*
  - *Alt sınıftaki override edilecek metodun erişim belirleyicisi, üst sınıftaki override edilen metodun erişim belirleyicisinden daha yüksek derecede olmalı.*

- **overloading ve overriding arasındaki fark:** *Overriding'de metod adı, parametreler, tipler aynı olmak zorundadır. Overloading'te parametre sırası, veya tipleri veya sayısından en az biri farklı olmalıdır.*

```java
package Kalitim;

class sA{
    int x, y;
    sA (int i, int j){ // Yapıcı
        x=i;
        y=j;
    }
    void yazdir() {
        System.out.println("x ve y değerleri: "
                +x+" "+y);
    }
}
class sB extends sA{
    int z;
    sB(int i, int j, int k){ // Yapıcı
        super(i,j); // i ve j sA'ya gönderildi
        z=k;
    }
    void yazdir(String yazi) {
        System.out.println(yazi + z);
    }

}
public class calistirOverriding2 {
    public static void main(String[] args){
        sB nesne = new sB(6,1,8);
        nesne.yazdir("Z'nin degeri: ");
        nesne.yazdir();
    }
}
```

Console | Problems | Debug Shell | Search

<terminated> calistirOverriding2 [Java Application] C:\Program Files\Java\jdk-13.(

```
Z'nin degeri: 8
x ve y değerleri: 6 1
```

- 1- İlk olarak main metodundan başlayalım. sB sınıfından nesne oluşturduk ve parantezler içerisinde sB'nin yapıcısına 3 değer gönderdik. Bu değerlerden ilk ikisi, super () ile üst sınıfın yapıcısına gönderildi. Gönderilen son değer ise z değişkenine atandı.
- 2- Daha sonra nesne.yazdir() diyerek parametre olarak bir String değeri gönderdik. Dolayısıyla sB sınıfındaki yazdir() metodu çalıştı. Normalde böyle bir metod olmasaydı doğrudan A sınıfındaki metod çalıştırılırdı. Fakat sB sınıfı diyor ki, ben sA'nın metodunu kullanmak istemiyorum kendi metodumu oluşturdum.
- 3- Son olarak nesne.yazdir() metodunu parametre göndermeden çalıştırdık. sB sınıfında böyle bir metod olmadığı için default olarak sA sınıfındaki yazdir () metodu çalıştırıldı.

FINAL ANAHTAR KELİMESİ

- **final** anahtar kelimesi Java'da temelde sabitler tanımlamaya yarar. (**Pi** sayısı vb.)
- **final** anahtar kelimesi sadece değişkenlere değil ayrıca sınıflara, metodlara ve parametrelere uygulanabilir.
  - Bir **sınıfı** final olarak tanımlarsak bu sınıftan türetme yapamayız.
  - Bir **metodu** final olarak tanımlarsak bu metodlar override edilemezler.
  - Bir **parametreyi** final olarak tanımlarsak bu parametrenin değerini değiştiremeyiz.
  - Bir **değişkeni** final olarak tanımlarsak bu değişkenin değerini değiştiremeyiz.

FINAL İLE SABİT TANIMLAMA

- final anahtar kelimesi İle tanımlanmış değişkenlerin değeri değiştirilemez. Bunlara sabit değişkenler denir. (Sabitler)
- final değişkenler ilk değeri atanmadan kullanılamazlar. Normalde bir değişken tanımlanıp değeri verilmediğinde default değere sahip olurdu. Fakat final değişkenlerde bu şekilde yaparsak hata ile karşılaşırız.

```
1  package Kalitim;
2
3  public class final_ {
4
5      final int x;
6
7      public static void main(String[] args) {
8          // TODO Auto-generated method stub
9
10     }
11
12 }
```

```
1  package Kalitim;
2
3  public class final_ {
4
5      final int x=10;
6
7      public static void main(String[] args) {
8          // TODO Auto-generated method stub
9
10     }
11
12 }
```

- final ile değişken tanımlamasında yanlış olan bir şey de değişkeni ilk değeri olmadan tanımlıyoruz ve başka bir yerde ona değer veriyoruz. Bu kullanım hatalıdır. Bu şekilde tanımlanmış değişkenlerin ilk değerleri, tanımlandığı ilk anda ya da yapıcılar içerisinde verilmelidir. Fakat metodlar içerisinde final tanımlarsak, başka satırda da ilk değeri verilebilir.

```
1  package Kalitim;
2
3  public class final_2 {
4      private final int x;
5
6      public final_2 (int deger) {
7          x=deger;
8      }
9      public void yaz() {
10         System.out.println(x);
11     }
12
13     public static void main(String[] args) {
14             final_2 nesne = new final_2(25);
15             nesne.yaz();
16     }
17 }
```

Console Problems Debug Shell Search

<terminated> final_2 [Java Application] C:\Program Files\Java\jdk-13.0.1\

25

- final anahtar kelimesi ile tanımlanmış değişkenlerin değeri, programın herhangi bir yerinde değiştirilemez.

```
1  package Kalitim;
2  public class final_3 {
3
4      private final int x;
5
6      public final_3 (int deger) {
7          x=deger;
8      }
9
10     public static void main(String[] args) {
11             final_3 nesne = new final_3(10);
12             System.out.println(nesne.x);
13             nesne.x=15;
14     }
15 }
```

Final ile Metodların Override Edilmesini Engelleme

- Metodları **final** olarak tanımlarsak, bu metodlar override edilemezler.

```
1  package Kalitim;
2
3  class final_Metod {
4      public final void metod(){
5          System.out.println("Bu metod override edilemez");
6      }
7  }
8   class sinif extends final_Metod{
9      public  void metod(){
10         System.out.println("Bu metod override edilmeye çalışıldı");
11     }
12  }
13
```

- final parametreleri, bir metoda gönderdiğimiz parametrelerin değişmesini istemediğimiz durumlarda kullanılır.

```
1  package Kalitim;
2
3  public class FinalParametre {
4      public void gonder(){
5          int x=5;
6          al(x);
7      }
8      public void al(final int y){
9          y=y+y;
10     }
11 }
```

```
1  package Kalitim;
2
3  public class FinalParametre {
4      public void gonder(){
5          int x=5;
6          al(x);
7      }
8      public void al(final int y){
9          int x=y*2;
10         y=y*2;
11     }
12 }
```

Final ile Sınıfların Kalıtımını Engelleme

- Final sınıflar'dan başka sınıflar türetilemez.

```
1  package Kalitim;
2
3  final class Sinif{
4
5  }
6
7   class final_Kalitim extends Sinif {
8      public static void main(String[] args) {
9          final_Kalitim nesne = new final_Kalitim();
10     }
11 }
12
```

```
1  package Kalitim;
2
3   class Sinif{
4
5  }
6
7   class final_Kalitim extends Sinif {
8      public static void main(String[] args) {
9          final_Kalitim nesne = new final_Kalitim();
10     }
11 }
12
```

Ödev-1

- **Sekil** isimli bir sınıf tanımlayın. Bu sınıf içerisinde **isim** ve **renk** alanları ile bu alanları ekrana yazdıran, **alanını** ve **çevresini** hesaplayan bir metod bulunsun. **Cember** sınıfı bu sınıfı miras alsın ve içerisinde yarıçap alanı bulunsun. **Sekil** sınıfının alan ve **çevre** hesabi yapan metotlarını **override** ederek yeniden bu metodu yazın. **Dikdortgen** sınıfı yine **Sekil** sınıfını miras alsın. Bu sınıfın **uzun kenar** ve **kısa kenar alanları** bulunsun. Bu sınıf da **Sekil** sınıfının **alan** ve **çevre** metodlarını **override** etsin. Son olarak **kare** sınıfı tanımlayın bu sınıf **Dikdortgen** sınıfını miras alsın ve **kenar alanı** bulunsun ve **dikdortgen** sınıfının metodlarını override etsin.

KAYNAKLAR

- KİRAZLI, M., ‘‘ Java 7:Yeni Başlayanlar için’’, Kodlab, 8.Baskı, Ekim,2015.